Matheus Avellar de Barros (117038497)
2020.★

Atividade 1

Acredito que, em uma implementação hipotética, o mais adequado (ou o que mais faria sentido, para mim) seria utilizar um **estilo baseado em eventos**. Ao receber uma mensagem, o servidor pode reagir e processar a essa mensagem recebida. O mesmo acontece com o cliente. É, ao meu entendimento, o estilo que estamos usando até então (com **recv** aguardando evento enviado pelo **send**).

1. **Componentes principais**:
    a. **Servidor**, para gerenciar as mensagens enviadas *por* clientes e *para* clientes;
    b. **Cliente**, que envia as mensagens para o servidor para serem encaminhadas para outros clientes.
2. **Ligações entre componentes** (conectores):
    Os clientes não comunicam-se entre si diretamente; somente comunicam-se com o servidor. Este é único, e não se comunica com outros servidores, somente com os clientes.
3. **Dados trocados entre componentes**:
    Serão trocados JSONs com as informações:
    a. ID do cliente que enviou a mensagem;
    b. ID do cliente destinatário;
    c. tipo de mensagem enviada;
    d. dados da mensagem (i.e. o texto ou informação).

Atividade 2

Uma arquitetura híbrida poderia ser construída, em que um servidor indica os endereços dos clientes para que eles conversem entre si; porém, acredito que para nossos propósitos, uma **arquitetura centralizada** (**cliente/servidor**) será mais que suficiente, e mais simples de implementar.

1. **Como serão agrupados**:
    Clientes utilizam o mesmo código, que é diferente do código do servidor. Ver 1.1 acima.
2. **Onde serão implantados**:
    Tanto os clientes quanto o servidor serão implantados em uma mesma máquina, pois nosso escopo aqui é mais próximo de um PoC/MVP do que um sistema complexo e robusto.
3. **Como deverão interagir**:
    Componentes irão interagir por via de JSONs contendo informações sobre o tipo de mensagem está sendo enviada (ex.: conectou-se, desconectou-se, enviou mensagem). O servidor funcionará como um servidor de echo, porém ao invés de devolver a mensagem para o mesmo cliente que lhe mandou, ele enviará para o destinatário.

Atividade 3

O protocolo (customizado) utilizado fará uso de **JSON**, como mencionado anteriormente.

1. **Tipos de mensagens**:

   As mensagens especificam seu tipo na propriedade `type` (ou equivalente). Por exemplo, quando um usuário conecta-se, a propriedade pode ter um valor de "new". Quando um usuário envia uma mensagem (de chat), pode ter um valor de "msg". Quando um usuário se desconecta, "out".
2. **Sintaxe/formato das mensagens**:

   As mensagens seguirão o seguinte formato:
   `{ id: <int>, to: <int>, type: <string>, text: <string> }`
   Como descrito em 1.3, acima:
   a. `id`: ID do cliente que enviou a mensagem;
   b. `to`: ID do cliente destinatário;
   c. `type`: tipo de mensagem enviada;
   d. `text`: dados da mensagem (i.e. o texto ou informação).
3. **Regras de envio/resposta**:

   Mensagens serão enviadas entre os componentes (e invisíveis aos usuários) da seguinte maneira:
   `<tamanho em bytes>,<JSON>`
   Dependendo do tempo de desenvolvimento disponível, talvez seja incluído um passo de compressão (por exemplo, com Huffman) antes do envio, tornando a mensagem mais curta e o envio mais rápido, da seguinte forma:
   `<tamanho comprimido em bytes>,<JSON comprimido>`
   Mensagens podem ser enviadas simplesmente utilizando a função `sendall`. Porém, para recebimento, o programa irá iterar o recebimento até que consiga consumir toda a mensagem (cujo tamanho é conhecido), como visto em aula.